EUROPEAN ASSOCIATION FOR MACHINE TRANSLATION

EAMT2018

Proceedings of the

21st Annual Conference of the European Association for Machine Translation

28–30 May 2018

Universitat d'Alacant

Alacant, Spain

Edited by

Juan Antonio Pérez-Ortiz

Felipe Sánchez-Martínez

Miquel Esplà-Gomis

Maja Popović

Celia Rico

André Martins

Joachim Van den Bogaert

Mikel L. Forcada

Organised by

Universitat d'Alacant

Universidad de Alicante

transducens

research group

ISBN: 978-84-09-01901-4

Deep Neural Machine Translation with Weakly-Recurrent Units

Mattia A. Di Gangi
FBK, Trento, Italy
University of Trento, Italy
digangi@fbk.eu

Marcello Federico
MMT Srl, Trento, Italy
FBK, Trento, Italy
federico@fbk.eu

Abstract

Recurrent neural networks (RNNs) have represented for years the state of the art in neural machine translation. Recently, new architectures have been proposed, which can leverage parallel computation on GPUs better than classical RNNs. Faster training and inference combined with different sequence-to-sequence modeling also lead to performance improvements. While the new models completely depart from the original recurrent architecture, we decided to investigate how to make RNNs more efficient. In this work, we propose a new recurrent NMT architecture, called Simple Recurrent NMT, built on a class of fast and weakly-recurrent units that use layer normalization and multiple attentions. Our experiments on the WMT14 English-to-German and WMT16 English-Romanian benchmarks show that our model represents a valid alternative to LSTMs, as it can achieve better results at a significantly lower computational cost.

1 Introduction

Neural machine translation (NMT) (Sutskever et al., 2014; Bahdanau et al., 2015) is a sequence-to-sequence problem that requires generating a sentence in a target language from a corresponding sentence in a source language. Similarly to other language processing task, NMT has mostly employed recurrent neural networks (RNNs) (Sennrich et al., 2016b; Sennrich et al., 2017b; Luong and Manning, 2015), in both their LSTM (Hochreiter and Schmidhuber, 1997) and GRU (Cho et al., 2014) variants, which can model long-range dependencies. Besides their simplicity, the choice of RNNs is also due to their expressive power, which has been proven equivalent to Turing Machines (Siegelmann and Sontag, 1995). RNNs have represented so far the state of the art of machine translation, and have constantly been enhanced to improve their performance. Nonetheless, their explicit time dependencies make training of deep RNNs computationally very expensive (Wu et al., 2016; Barone et al., 2017).

Recent works have proposed new NMT architectures, not based on RNNs, that obtained significant improvements both in training speed and translation quality: the so-called convolutional sequence-to-sequence (Gehring et al., 2017) and the self-attentive or transformer (Vaswani et al., 2017) models. Speed improvements by these models mainly come from the possibility of parallelizing computations over word sequences, as both models do not have time dependencies. On the other hand, performance improvements appear to be due to the path lengths needed by the networks to connect distant words in a sentence: linear for RNNs, logarithmic for convolutional models, and constant for the transformer.

In this paper we propose a neural architecture that shares some properties with the above-mentioned models, while maintaining a recurrent design. Our hypothesis is that current RNNs for NMT have not been designed to take full advantage of deep structures and that better design could lead to improved performance and efficiency. Contemporary to this work, Chen et al. (2018) have shown that RNN can still outperform the transformer model when using better hyper-parameters.

Pérez-Ortiz, Sánchez-Martínez, Esplà-Gomis, Popović, Rico, Martins, Van den Bogaert, Forcada (eds.)
Proceedings of the 21st Annual Conference of the European Association for Machine Translation, p. 119–128
Alacant, Spain, May 2018.

We start by discussing previous efforts that proposed simplified and theoretically grounded versions of the LSTM RNN, which very recently lead to the so-called Simple Recurrent Unit (SRU). Then, we introduce our NMT architecture based on weakly-recurrent units, which we name Simple Recurrent NMT (SR-NMT). We present machine translation results on two public benchmark, WMT14 English-German and WMT16 English-Romanian, and compare the results of our architecture against LSTM and SRU based NMT, using similar settings for all of them. Results show that SR-NMT trains faster than LSTM NMT and outperforms both LSTM and SRU NMT. In particular, SR-NMT with 8-layers even outperforms Google's NMT 8-layer LSTM architecture (Wu et al., 2016). Moreover, training our model took the equivalent of 12 days on a single K80 GPU against the 6 days on 96 K80 GPUs reported by (Wu et al., 2016). Finally, the NMT architecture presented in this paper was developed in OpenNMT-py (Klein et al., 2017) and the code is publicly available on Github[1].

2 Related works

RNNs are an important tool for NMT, and have ranked at the top of the WMT news translation shared tasks (Bojar et al., 2017) in the last three years (Luong and Manning, 2015; Sennrich et al., 2016b; Sennrich et al., 2017b). Recurrent NMT is also the first approach that outperformed phrase-based statistical MT (Bentivogli et al., 2016). Despite the important results, training of RNNs remains inefficient because of an intrinsic lack of parallelism and the necessity of redundant parameters in its LSTMs and GRUs (Ravanelli et al., 2018; Zhou et al., 2016) variants. Sennrich et al. (2017b) reduce training time in two different ways: by reducing the network size with tied embeddings (Press and Wolf, 2017) and by adding layer normalization to their architecture (Ba et al., 2016). In fact, the reduction of the covariate shift produced by this mechanism shows to significantly speed up convergence of the training algorithm. Of course, it does not alleviate the lack of parallelism.

Pascanu et al. (2014) studied RNNs and found that the classical stacked RNN architecture does not have a clear notion of *depth*. In fact, when performing back-propagation through time, the gradient is sent backward in both the horizontal and vertical dimensions, thus having a double notion of depth, which also hurts the optimization procedure. They propose as a solution the notions of *deep transition*, from one hidden state to the following hidden state, and the notion of *deep output*, from the last RNN layer to the network output layer. The winning model in WMT17 actually implemented both of them (Sennrich et al., 2017b; Sennrich et al., 2017a).

Balduzzi and Ghifary (2016) proposed strongly-typed RNNs, which are variants of vanilla RNN, GRU and LSTM that respect some constraints and are theoretically grounded on the concept of *strongly-typed quasi-linear algebra*. A strongly-typed quasi-linear algebra imposes constraints on the allowed operations for an RNN. In particular, in this framework there is a constraint inspired from the type system from physics, and one inspired by functional programming. The idea of types forbids the sum of vectors generated from different branches of computation. In the case of RNNs, this means that it is not possible to sum among them the previous hidden state and the current input, as they are produced by different computation branches. The second constraint aims to simulate the distinction among *pure functions* and functions with side effects, typical of functional programming. In fact, as RNNs own a state, they can approximate *algorithms* and also produce "side effects". According to the authors, side effects manifest when the horizontal (time-dimension) connections are altered, and are the reason behind the poor behavior of techniques such as dropout (Srivastava et al., 2014) or batch normalization (Ioffe and Szegedy, 2015) when they are applied to the horizontal direction straightforwardly (Laurent et al., 2016; Zaremba et al., 2014). Thus, the side effects should be confined to a part of the network that cannot hinder the learning process. The solution they propose consists in using learnable parameters only in stateless equations (*learnware*), while the states are combined in parameterless equations (*firmware*). The combination is achieved through the use of *dynamic average pooling* (or peephole connections), which allows the network to use equations with parameters to compute the states and the gates, and then use the gate vectors to propagate forward horizontally the hidden state. The authors show theoretically that strongly-typed RNNs have generalization capabilities similar to their classical coun-

[1] https://github.com/mattiadg/SR-NMT

terparts, and confirm it with an empirical investigation over several tasks, where the strongly-typed RNNs achieve results not worse than their classical counterparts while training for less time. In addition, the absence of parameters in the state combination cancels the problem of depth introduced by Pascanu and colleagues, as these models need only the classical back-propagation and not back-propagation through time.

Quasi-recurrent neural networks (Bradbury et al., 2017) are an extension of the previous work that use gated convolutions in order to not compute functions of isolated input tokens, but always consider the context given by a convolutional window.

SRUs (Lei et al., 2017b), are a development of the units proposed by Balduzzi and Ghifary designed for training speed efficiency. The equations can be easily CUDA optimized, while a good task performance is obtained by stacking many layers in a deep network. SRUs use *highway connections* (Srivastava et al., 2015) to enable the training of deep networks. Moreover, SRUs can parallelize the computation over the time steps also in the decoder. In fact during training the words of the whole sequence are known and there is no dependency on the output of the previous time step. As for strongly-typed RNNs, the information from the context is propagated with dynamic average pooling, which is much faster to compute than matrix multiplications. SRUs were tested on a number of tasks, including machine translation, and showed performance similar to LSTMs, but with significantly lower training time. However to obtain results comparable to a weak LSTM-based NMT, SRUs require many more layers of computation. The results show that a single SRU has a significantly lower representation capability than a single LSTM. In addition, every layer adds little overhead in terms of training time per epoch, but also the results show little improvement.

In this work we further develop the idea of SRUs, and propose an NMT architecture that can outperform LSTM-based NMT.

3 Simple Recurrent NMT

We propose a sequence-to-sequence architecture that uses an enhanced version of SRUs (see Figure 1) to improve the training process, in particular with many layers, and increase the representation capability. In fact, although Lei et al. (2017b) show

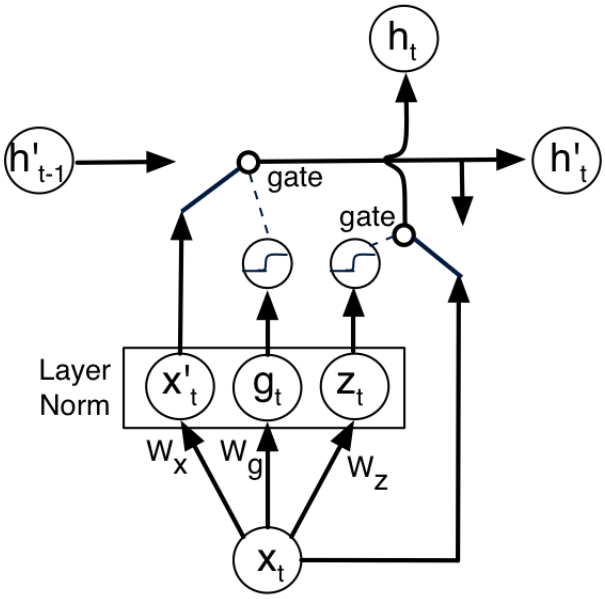

Figure 1: Core weakly-recurrent unit used in the SR-NMT architecture. Layer normalization is performed only once for all the transformations. At the end of the unit, the gate $\mathbf{z}_t$ is used for the highway connection.

that they can train networks with up to 10 layers of SRUs, both in encoder and decoder, without overfitting, their results are far from the state of the art of recurrent NMT. Our design goals are addressed in a way similar to (Gehring et al., 2017; Vaswani et al., 2017). We add an attention layer within every decoder unit, and make the training more stable by adding a layer normalization layer (Ba et al., 2016) after every matrix multiplication with a parameter matrix. The layer normalization reduces the covariate shift (Ioffe and Szegedy, 2015), thus it makes easier the training of deep networks. In addition to layer normalization, our units use highway connections (Srivastava et al., 2015), which enable the training of deep networks. Our SR-NMT architecture is shown in Figure 2

Our weakly-recurrent units used in the encoder and decoder both separate *learnware* and *firmware*, although not being strongly typed (Balduzzi and Ghifary, 2016) as they include highway connections summing vectors of different types. In the following, we introduce in detail the encoder and decoder networks of our simple recurrent NMT architecture.

3.1 Encoder

Our encoder uses bidirectional weakly-recurrent units with layer normalization. We use two candidate hidden states ($\overrightarrow{\mathbf{h}}_i$, $\overleftarrow{\mathbf{h}}_i$) and two recursion

Model	train speed
LSTM 2L	3700 tok/s
SRU 3L	4600 tok/s
SR-NMT 1L	7900 tok/s
SR-NMT 2L	5500 tok/s
SR-NMT 3L	4300 tok/s
SR-NMT 4L	3600 tok/s

Table 1: Training speed comparison of our architectures with LSTM and SRU baselines on WMT14 En-De. Timings are performed on a single Nvidia Gtx 1080 GPU with CUDA 8.0 and pytorch 0.2.

gates ($\overrightarrow{\mathbf{g}}_i, \overleftarrow{\mathbf{g}}_i$) for the two directions. The candidate hidden state for every time step is computed as a weighted average among the current input and the previous hidden state, controlled by the two gates (peephole connections). We apply a single normalization (*LN*) for each layer to improve training convergence and impose a soft constraint among the parameters. Finally, the input of each layer is combined with its output through highway connections. Formally, our encoder layer is defined by the following equations:

$$\mathbf{x_i} \in R^d; \quad \mathbf{W} \in R^{d \times (4\frac{d}{2}+d)}$$

$$[\overrightarrow{\mathbf{x}}_i, \overleftarrow{\mathbf{x}}_i, \overrightarrow{\mathbf{g}}_i, \overleftarrow{\mathbf{g}}_i, \mathbf{z_i}] = LN(\mathbf{x_i}\mathbf{W})$$

$$\begin{aligned}
\overrightarrow{\mathbf{h}}_i &= (1-\sigma(\overrightarrow{\mathbf{g}}_i)) \odot \overrightarrow{\mathbf{h}}_{i-1} + \sigma(\overrightarrow{\mathbf{g}}_i)) \odot \overrightarrow{\mathbf{x}}_i \\
\overleftarrow{\mathbf{h}}_i &= (1-\sigma(\overleftarrow{\mathbf{g}}_i)) \odot \overleftarrow{\mathbf{h}}_{i+1} + \sigma(\overleftarrow{\mathbf{g}}_i) \odot \overleftarrow{\mathbf{x}}_i \\
\mathbf{h}_i &= (1-\sigma(\mathbf{z}_i)) \odot [\overrightarrow{\mathbf{h}}_i; \overleftarrow{\mathbf{h}}_i] + \sigma(\mathbf{z}_i) \odot \mathbf{x}_i
\end{aligned}$$

3.2 Decoder

The decoder employs unidirectional units, with layer normalization (*LN*) after every matrix multiplication similarly to the encoder units, and has an attention mechanism in every layer. The attention output is combined with the layer's hidden state in a way similar to the *deep output* (Pascanu et al., 2014) used by Luong (2015). The highway connection is applied only at the end of the unit. The presence of multiple attention models connected to the last encoder layer produces a high gradient for the encoder output, thus we scale the gradient dividing the attention output by $\sqrt{d}$. This kind of scaling has been proposed in (Vaswani et al., 2017) inside the transformer model, but we observed empirically that this version works better for our model. Formally:

$$\mathbf{y_i} \in R^d; \;\; \mathbf{W} \in R^{d \times 3d}; \;\; \mathbf{W_s}, \mathbf{W_c} \in R^{d \times d}$$

$$\begin{aligned}
[\tilde{\mathbf{y}}_\mathbf{i}, \mathbf{g_i}, \mathbf{z_i}] &= LN(\mathbf{y_i}\mathbf{W}) \\
\tilde{\mathbf{s}}_\mathbf{i} &= (1-\sigma(\mathbf{g_i})) \odot \tilde{\mathbf{s}}_{\mathbf{i-1}} + \sigma(\mathbf{g_i}) \odot \tilde{\mathbf{y}}_\mathbf{i} \\
\mathbf{c_i} &= attn(\tilde{\mathbf{s}}_\mathbf{i}, \mathbf{H})(1/\sqrt{d}) \\
\mathbf{o_i} &= \tanh(LN(\tilde{\mathbf{s}}_\mathbf{i}\mathbf{W_s}) + LN(\mathbf{c_i}\mathbf{W_c})) \\
\mathbf{s_i} &= (1-\sigma(\mathbf{z_i})) \odot \mathbf{o_i} + \sigma(\mathbf{z_i}) \odot \mathbf{y_i}
\end{aligned}$$

The decoder includes a standard *softmax* layer over the target vocabulary which is omitted from this description. For our architecture, we opted for a layer-normalized version of the MLP global attention (Bahdanau et al., 2015), which showed to perform better than the dot attention model (Luong et al., 2015):

$$\begin{aligned}
\tilde{\alpha_{\mathbf{ij}}} &= \mathbf{v}_\alpha \tanh(LN(\tilde{\mathbf{s}}_\mathbf{i}\mathbf{W_{as}}) + LN(\mathbf{h_j}\mathbf{W_{ah}})) \\
\boldsymbol{\alpha_i} &= \text{softmax}(\tilde{\boldsymbol{\alpha}}_i) \\
\mathbf{c_i} &= \sum_{i=0}^{L} \alpha_{ij}\mathbf{h_j}
\end{aligned}$$

Our SR-NMT architecture stacks several layers both on the encoder and decoder sides, as shown in Figure 2. The natural structure we consider is one having the same number of layers on both sides, although different topologies could be considered, too.

4 Experiments

We implemented our architecture in PyTorch (Paszke et al., 2017) inside the OpenNMT-py toolkit (Klein et al., 2017). All the tested models have been trained with the Adam (Kingma and Ba, 2015) optimizer until convergence, using the typical initial learning rate of 0.0003, and default values for β_1 and β_2. At convergence, the models were further trained until new convergence with learning rate 0.00015 (Bahar et al., 2017). The model used to restart the training is selected according to the perplexity on the validation set. We applied dropout of 0.1 before every multiplication by a parameter matrix, and in the case of LSTM it is applied only to vertical connections in order to use the LSTM version optimized in CUDA. The batch size is 64 for all the experiments and all the layers for all the models have an output size of 500.

4.1 Datasets

We used as benchmarks the WMT14 English to German and the WMT16 English to Romanian

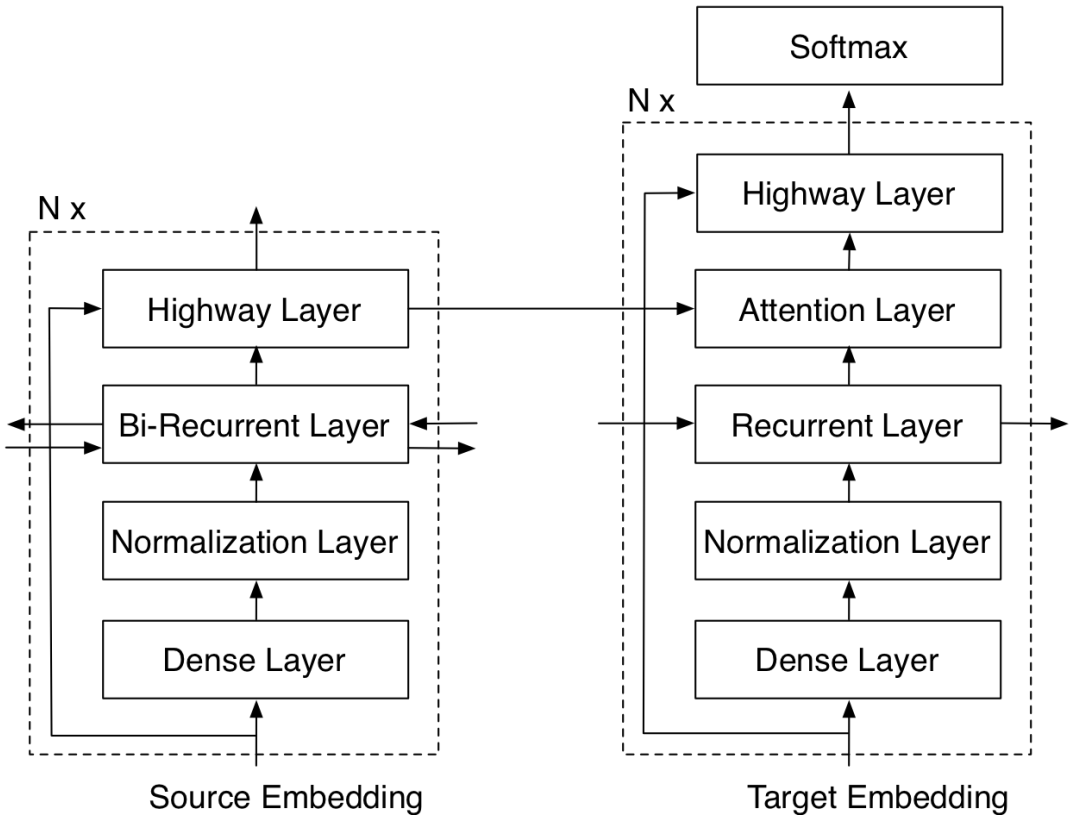

Figure 2: SR-NMT encoder-decoder architecture. On the left, a single encoder block to repeat N times. The output of the last layer is used as input for the decoder's attention layers. On the right, a decoder block to repeat N times. The first three sub-layers are the same in the encoder and the decoder, but the latter has an attention layer before the highway connection.

datasets.

In the case of WMT14 En-De, the training set consists of the concatenation of all the training data that were available for the 2014 shared task, the validation set is the concatenation of newstest2012 and 2013, and newstest2014 is our test set. Then, it was preprocessed with tokenization, punctuation normalization and de-escaping of the special characters. Furthermore, we applied BPE segmentation (Sennrich et al., 2016a) with 32,000 merge rules. We removed from the training data all the sentence pairs where the length of at least one sentence exceeded 50 tokens, resulting on a training set of $3.9M$ sentence pairs. Furthermore, we cleaned the training set by removing sentences in a wrong language and poorly aligned sentence pairs. For the cleaning process we used the automatic pipeline developed by the ModernMT project[2].

In the case of WMT16 En-Ro, we have used the same data and preprocessing used by Sennrich et al. (2016b) and Gehring et al. (2017). The back-translations to replicate the experiments are available[3] and we applied the same preprocessing[4], which involves punctuation normalization, tokenization, truecasing and BPE with $40K$ merge rules.

5 Evaluation

In this section, we describe the evaluation of our models with the two benchmarks. As our main goal is to prove that SR-NMT represent a valid alternative to LSTMs, we have put more effort on WMT14 En-De, which is widely used as a benchmark dataset. The experiments on WMT16 En-Ro are aimed to verify the effectiveness of our models in a different language pair with a different data size.

5.1 WMT14 English to German

The results for WMT14 En-De are evaluated on cased output, tokenized with the tokenizer script from the Moses toolkit (Koehn et al., 2007), and the BLEU score is computed using multi-bleu.pl

[2]https://github.com/ModernMT/MMT

[3]http://data.statmt.org/rsennrich/wmt16_backtranslations/en-ro .

[4]https://github.com/rsennrich/wmt16-scripts/blob/80e21e5/sample/preprocess.sh

from the same toolkit. With this procedure the results are comparable with the results reported from the other publications[5].

We compare our models with the results reported in (Lei et al., 2017b), and also reproduce some of their experiments. We train 4 baseline models following the same procedure used for SR-NMT. Three baselines are LSTM-based NMT models as provided by OpenNMT-py, with 2, 3 and 5 layers in both encoder and decoder. The other is an SRU model with 3 layers that we re-implemented in PyTorch, in order to perform a more fair comparison with our model. For the baselines we use dropout after every layer and MLP attention (Luong et al., 2015), both resulting in better results than the default implementation. Furthermore, we compare our results with Google's NMT system (Wu et al., 2016), Convolutional S2S model (Gehring et al., 2017), and the Transformer (Vaswani et al., 2017).

5.2 WMT16 English to Romanian

In the case of English to Romanian, we trained our models with the same hyper-parameters used for English to German, despite the difference in the amount of data. The BLEU score is computed using the official script of the shared task[6], which runs on cased and detokenized output.

We did not implement baselines for this language pair, and we compare our results with the winning submission of the WMT16 shared task (Sennrich et al., 2016b), with the Convolutional S2S model (Gehring et al., 2017) and the Transformer (Gu et al., 2018).

WMT14 En-De	**BLEU**	**# par**
LSTM 2L	21.82	62M
LSTM 3L	22.26	65M
LSTM 5L	22.72	72M
SRU 3L	20.88	59M
SR-NMT 1L	18.33	56M
SR-NMT 2L	21.82	58M
SR-NMT 3L	22.35	61M
SR-NMT 4L	23.32	63M
SR-NMT 5L	24.11	66M
SR-NMT 6L	23.93	68M
SR-NMT 7L	24.34	71M
SR-NMT 8L	24.87	73M
SR-NMT 9L	25.04	76M
SR-NMT 10L	24.98	78M
Setting of (Lei et al., 2017b)		
LSTM 2L	19.67	84M
LSTM 5L	20.45	96M
SRU 3L	18.89	81M
SRU 10L	20.70	91M
GNMT (Wu et al., 2016)		
LSMT 8L	24.61	-
Ensemble	26.30	-
Convolutional (Gehring et al., 2017)		
ConvS2S 15L	25.16	-
Ensemble	26.43	-
Transformer (Vaswani et al., 2017)		
Base 6L	27.30	65M
Big 6L	**28.40**	213M

Table 2: Experiments with cleaned data on WMT14 En-De both for our architectures and the baselines, and comparison with the state of the art.

6 Results

In this section, we discuss the performance in terms of training speed and translation quality of our architecture.

6.1 WMT14 En-De

In the first part of Table 2 we list the results of SR-NMT using from 1 up to 10 layers and our baselines. The training speeds are reported in Table 1.

SR-NMT with 3 layers has a number of parameters comparable to the LSTM baseline with 2 layers, but its training speed is 14% faster (4300 tok/s vs 3700 tok/s), and the BLEU score is 0.5 points higher. Moreover, the implementation of the LSTM is optimized at CUDA level, while our architecture is fully implemented in PyTorch and could be made faster following the optimizations of Lei et al. (2017b). Furthermore, also the layer normalization can be implemented faster in CUDA[7]. By increasing the number of LSTM layers from 2 to 5, the improvement in terms of BLEU score is only 0.9 points, and it is worse than SR-NMT with 4 layers.

The comparison with NMT based on SRUs is in favor of our architecture, which achieves higher translation quality with less layers. In particular, SR-NMT with 2 layers outperforms SRU NMT with 3 layers by 1 BLEU point and also trains

[5] https://github.com/tensorflow/tensor2tensor/blob/master/tensor2tensor/utils/get_ende_bleu.sh

[6] mteval-v13a.pl

[7] https://github.com/MycChiu/fast-LayerNorm-TF

WMT16 En-Ro	BLEU
SR-NMT 1L	24.74
SR-NMT 2L	26.41
SR-NMT 4L	28.81
SR-NMT 6L	29.04
SR-NMT 8L	28.70
GRU (Sennrich et al., 2016b)	
GRU 1L+2L	28.1
Ensemble	28.2
Convolutional (Gehring et al., 2017)	
ConvS2S 15L	30.02
Transformer (Gu et al., 2018)	
NAT	29.79
Transformer	**31.91**

Table 3: Results on the test set of WMT16 En-Ro and comparison with the state of the art.

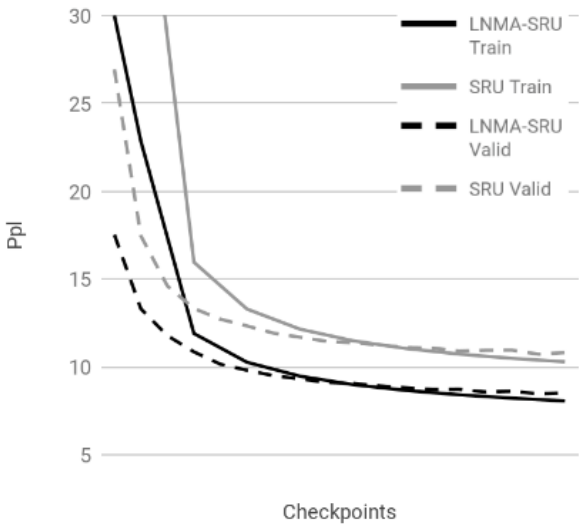

Figure 3: Perplexity against time for SR-NMT and SRU-based NMT with 3 layers and the same optimization policy. The convergence is achieved after a comparable number of iterations, but SR-NMT achieves a better convergence point.

faster (Table 1). However, this comparison is performed with implementations that are not optimized for fast execution in GPU. A speed comparison with optimized implementations could lead to different results.

In the second part of Table 2 we report some results from (Lei et al., 2017b) on the same benchmark. The different number of parameters is probably due to a different size of the vocabulary, in fact the number of merge rules used is not reported in the paper. Our LSTM baseline performs clearly better then the one cited because of the straightforward improvements we implemented, i.e. the use of *input feeding* (Luong et al., 2015), MLP attention instead of general or dot attention, and dropout in every layer. With this improvements, our baseline with 2 layers obtains 1.4 BLEU scores more than its counterpart using 5 layers. Moreover, it also outperforms SRU with 10 layers by more than 1 point. This result shows that our additions are fundamental to have a competitive architecture based on weakly recurrent units.

Figure 3 shows a comparison of the learning curves of SR-NMT and SRU NMT both with 3 layers. We can easily observe that the convergence of SR-NMT occurs at comparable speed but to a better point and the validation perplexities of the two models are very close to the training perplexities.

When we compare SR-NMT to GNMT (Table 2), we can observe that SR-NMT with 8 layers performs slightly better than GNMT, which in turn uses many more parameters, as it uses 8 LSTM layers with size 1024. Moreover, GNMT was trained for 6 days on 96 Nvidia K80 GPUs, while our model took the equivalent[8] of 12 days on a single K80 GPU.

Our best BLEU score, 25.04, is obtained with 9 layers. This is only 0.12 BLEU points below the convolutional model that used 15 layers in both encoder and decoder, and hidden sizes of at least 512. Finally, we notice that SR-NMT's best performance is still below that of the transformer model. Future work will be devoted to deeper explore the hyper-parameter space of our architecture and enhance it along the recent developments in (Chen et al., 2018).

6.2 WMT16 En-Ro

The results for WMT16 En-Ro are listed in Table 3. We obtain the highest score for this dataset with 6 layers, which can be due to the smaller dimension of the dataset, for which we did not add any form of regularization.

Our best SR-NMT system, which obtained a BLEU score of 29.04, is 1 BLEU point lower than ConvS2S, and almost 3 BLEU points lower than the state of the art. Nonetheless, this score is almost 1 BLEU point better than the score obtained by the winning system in WMT16 (Bojar et al., 2016), showing that SR-NMT represent a viable alternative to more complex RNNs.

[8] As our training currently works on single GPU, we could only fit models up to 7 layers into a K80, hence the estimate. Actually, models above 7 layers were trained on a V100 GPU.

Model	BLEU	Δ
LNMA-SRU 4L	22.99	0
- LayerNorm	21.97	-1.02
- Multi Attention	21.57	-1.42
- Highway	20.85	-2.14
- Ln & MA	20.51	-2.48
- LN & highway	/	- /
- MA & highway	19.54	-3.45
- LN, MA & highway	18.39	-4.6

Table 4: Ablation experiments on SR-NMT with 4 layers. BLEU scores are computed after one training stage. While removing multi attention we still keep one attention model in the last layer. The system without layer normalization and highway connections failed to converge.

7 Ablation experiments

In this section, we evaluate the importance of our enhancements to the original SRU unit, namely multi-attention and layer normalization, and of the highway connections, which were already present in the original formulation of SRUs.

We take our SR-NMT model with 4 layers and remove from it one component or a set of components. All the combinations are reported. Results refer to the WMT14 En-De task after performing only one training stage. In other words, we did not restart training after convergence as we did for the systems reported in Table 2. As our previous experiments already proved the superiority of SR-NMT to LSTMs, the goal of this section is to understand whether all the proposed additions are important and to quantify their contributions.

From Table 4 we can observe that the removal of highway connections causes the highest drop in performance (-2.14 BLEU points), followed by multi attention and then layer normalization. Another important observation is the additivity of the contributions from all the components, in fact when two or three components are removed at once, the drop in performance is roughly the sum of the drops caused by the single components. Finally, the removal of layer normalization and highway connections, while keeping multi attention, causes a gradient explosion that prevents the SR-NMT system from converging.

8 Conclusions

In this paper we have presented a simple recurrent NMT architecture that enhances previous SRUs (Lei et al., 2017b) by adding elements of other architectures, namely layer normalization and multiple attentions. Our goal was to explore the possibility to make weakly-recurrent units competitive with LSTMs for NMT. We have shown that our SR-NMT architecture is able to outperform more complex LSTM NMT models on two public benchmarks. In particular, SR-NMT performed even better than the GNMT system, while using a simpler optimization policy, a vanilla beam search and a fraction of its computational resources for training. Future work will be in the direction of further enhancing SR-NMT by integrating core components that seem to particularly boost performance of the best non recurrent NMT architectures.

Acknowledgements

We would like to thank Nicola Bertoldi for his technical support and the anonymous reviewers for their useful comments. We gratefully acknowledge the support of NVIDIA Corporation with the donation of GPUs used for this research.

References

Ba, Jimmy Lei, Jamie Ryan Kiros Geoffrey E. Hinton. 2016. Layer normalization. arXiv preprint arXiv:1607.06450.

Bahar, Parnia, Tamer Alkhouli, Jan-Thorsten Peter, Christopher J. S. Brix, and Hermann Ney. 2017. Empirical investigation of optimization algorithms in neural machine translation. *The Prague Bulletin of Mathematical Linguistics, 108(1)*, 13-25.

Bahdanau, Dzmitry, Kyunghyun Cho and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate *Proceedings of the 3rd International Conference on Learning Representations,* San Diego, USA.

Balduzzi, David and Muhammad Ghifary. 2016. Strongly-Typed Recurrent Neural Networks. *International Conference on Machine Learning*, 1292–1300.

Barone, Antonio Valerio Miceli, Jindrich Helcl, Rico Sennrich, Barry Haddow and Alexandra Birch. 2017. Deep architectures for Neural Machine Translation. *Proceedings of the Second Conference on Machine Translation*, Copenhagen, Denmark. 99-107.

Bentivogli, Luisa, Arianna Bisazza, Mauro Cettolo, Marcello Federico. 2016. Neural versus Phrase-Based Machine Translation Quality: a Case Study. *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, November 1-5, 2016, Austin, Texas, USA. 257–267

Bradbury, James, Stephen Merity, Caiming Xiong and Richard Socher. 2017. Quasi-Recurrent Neural Networks. *Proceedings of the 5th International Conference on Learning Representations*, Toulon, France.

Bojar, Ondřej, Rajen Chatterjee, Christian Federmann, Yvette Graham, Barry Haddow, Matthias Huck, Antonio Jimeno Yepes, Philipp Koehn, Varvara Logacheva, Christof Monz, Matteo Negri, Aurelie Neveol, Mariana Neves, Martin Popel, Matt Post, Raphael Rubino, Carolina Scarton, Lucia Specia, Marco Turchi, Karin Verspoor, Marcos Zampieri. 2016. Findings of the 2016 conference on machine translation (wmt16). *Proceedings of the First Conference on Machine Translation*, Berlin, Germany. 131–198.

Bojar, Ondřej, Rajen Chatterjee, Christian Federmann, Yvette Graham, Barry Haddow, Shujian Huang, Matthias Huck, Philipp Koehn, Qun Liu, Varvara Logacheva, Christof Monz, Matteo Negri, Matt Post, Raphael Rubino, Lucia Specia, Marco Turchi. 2017. Findings of the 2017 conference on machine translation (wmt17). *Proceedings of the Second Conference on Machine Translation*, Copenhagen, Denmark. 169–214.

Cettolo, Mauro, Christian Girardi and Marcello Federico. 2012. Wit3: Web inventory of transcribed and translated talks. *Proceedings of the 16th Annual Conference of the European Association for Machine Translation*, Trento, Italy.

Chen, Mia Xu, et al. 2018. The Best of Both Worlds: Combining Recent Advances in Neural Machine Translation. *arXiv preprint arXiv:1804.09849.*

Cho, Kyunghyun, Bart Van Merriënboer, Dzmitry Bahdanau and Yoshua Bengio. 2014. On the properties of neural machine translation: Encoder-decoder approaches *Syntax, Semantics and Structure in Statistical Translation (2014): 103.*

Gal, Yarin and Zoubin Ghahramani. 2016. A theoretically grounded application of dropout in recurrent neural networks. *In Advances in neural information processing systems*, 1019–1027.

Gehring, Jonas, Michael Auli, David Grangier, Denis Yarats and Yann N. Dauphin. 2017. Convolutional Sequence to Sequence Learning. *Proceedings of the 34th International Conference on Machine Learning*, Sidney, Australia. 1243–1252.

Gu, Jiatao, James Bradbury, Caiming Xiong, Victor O.K. Li, Richard Socher. 2018. Non-Autoregressive Neural Machine Translation. *Proceedings of the Sixth International Conference on Learning Representations (ICLR)*, Vancouver, Canada.

Hochreiter, Sepp and Jürgen Schmidhuber. 1997. Long short-term memory *Neural computation.* MIT Press 1735–1780.

Ioffe, Sergey and Christian Szegedy. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *International conference on machine learning*, 448–456.

Kalchbrenner, Nal, Lasse Espeholt, Karen Simonyan, Aaron van den Oord, Alex Graves and Koray Kavukcuoglu. 2016 Neural machine translation in linear time. *arXiv preprint arXiv:1610.10099.*

Kingma, Diederik P. and Jimmy Ba. 2015. Adam: A method for stochastic optimization. *3rd International Conference for Learning Representations*, San Diego, USA.

Klein, Guillaume, Yoon Kim, Yuntian Deng, Jean Senellart and Alexander M. Rush. 2017. *OpenNMT: Open-Source Toolkit for Neural Machine Translation. Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (ACL)*, System Demonstrations, 67-72.

Koehn, Philipp, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, Chris Dyer, Ondej Bojar, Alexandra Constantin, Evan Herbst. 2007. Moses: Open source toolkit for statistical machine translation. *Proceedings of ACL on interactive poster and demonstration sessions*, 177–180.

Laurent, César, Gabriel Pereyra, Philémon Brakel, Ying Zhang and Yoshua Bengio. 2016. Batch normalized recurrent neural networks. *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2657–2661.

Lei, Tao and Wengong Jin, Regina Barzilay and Tommi Jaakkola. 2017. Deriving Neural Architectures from Sequence and Graph Kernels. *Proceedings of the 34th International Conference on Machine Learning*, Sidney, Australia. 2024–2033.

Lei, Tao, Yu Zhang and Yoav Artzi. 2017. Training RNNs as Fast as CNNs. *arXiv preprint arXiv:1709.02755*.

Luong, Thang, Hieu Pham, and Christopher D. Manning. 2015. Effective Approaches to Attention-based Neural Machine Translation. *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing.* 1412–1421.

Luong, Minh-Thang and Christopher D. Manning. 2015. Stanford neural machine translation systems for spoken language domains. *Proceedings of the 12th International Workshop on Spoken Language Translation*, Da Nang, Vietnam. 76–79.

Papineni, Kishore, Salim Roukos, Todd Ward and Wei-Jing Zhu. 2002. BLEU: a Method for Automatic Evaluation of Machine Translation *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL).* 311–318.

Pascanu, Razvan, Tomas Mikolov and Yoshua Bengio. 2013. On the difficulty of training recurrent neural networks. *Proceedings of the 30th International Conference on Machine Learning*, Atlanta, USA. 1310–1318.

Pascanu, Razvan, Caglar Gulcehre, Kyunghyun Cho and Yoshua Bengio. 2014. How to construct deep recurrent neural networks. *Proceedings of the 2nd International Conference on Learning Representations*, Banff, Canada.

Paszke, Adam, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga and Adam Lerer. 2017. Automatic differentiation in PyTorch. *NIPS 2017 Autodiff Workshop*, Long Beach, USA.

Press, Ofir and Lior Wolf. 2017. Using the Output Embedding to Improve Language Models. *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics*: Volume 2, Short Papers (Vol. 2, pp. 157-163).

Ravanelli, Mirco, Philemon Brakel, Maurizio Omologo, and Yoshua Bengio. 2018. Light Gated Recurrent Units for Speech Recognition. *IEEE Transactions on Emerging Topics in Computational Intelligence 2*, no. 2 (2018): 92-102.

Sennrich, Rico, Barry Haddow, and Alexandra Birch. 2016. Neural Machine Translation of Rare Words with Subword Units. *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, Berlin, Germany. 1715–1725.

Sennrich, Rico, Barry Haddow and Alexandra Birch. 2016. Edinburgh Neural Machine Translation Systems for WMT 16. *Proceedings of the First Conference on Machine Translation: Volume 2, Shared Task Papers*, Vol. 2, 371–376.

Sennrich, Rico, Orhan Firat, Kyunghyun Cho, Alexandra Birch, Barry Haddow, Julian Hitschler, Marcin Junczys-Dowmunt, Samuel Lubli, Antonio Valerio Miceli Barone, Jozef Mokry, Maria Ndejde. 2017. Nematus: a Toolkit for Neural Machine Translation. *Proceedings of the Software Demonstrations of the 15th Conference of the European Chapter of the Association for Computational Linguistics*, Valencia, Spain. 65–68.

Sennrich, Rico, Alexandra Birch, Anna Currey, Ulrich Germann, Barry Haddow, Kenneth Heafield, Antonio Valerio Miceli Barone and Philip Williams. 2017. The University of Edinburgh's Neural MT Systems for WMT17. *Proceedings of the Second Conference on Machine Translation*, Copenhagen, Denmark. 389–399.

Siegelmann, Hava T. and Eduardo D. Sontag. 1995. On the computational power of neural nets. *Journal of computer and system sciences*, 50(1), 132-150.

Srivastava, Rupesh Kumar, Klaus Greff, and Jürgen Schmidhuber. 2015. Highway networks. ICML 2015 Deep Learning Workshop.

Srivastava, Nitish, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever and Ruslan Salakhutdinov. 2014 Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, Vol. 15, n. 1, 1929–1958.

Sutskever, Ilya, Oriol Vinyals and Quoc V. Le. 2014. Sequence to sequence learning with neural networks. *Advances in neural information processing systems.* 3104–3112.

Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jacob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser and Illia Polosukhin. 2017. Attention is all you need. In Advances in Neural Information Processing Systems (pp. 6000-6010).

Wu, Yonghui,Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, ukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes and Jeffrey Dean. 2016. Googles Neural Machine Translation System: Bridging the Gap between Human and Machine Translation. *arXiv preprint arXiv:1609.08144*

Zaremba, Wojciech, Ilya Sutskever and Oriol Vinyals. 2014. Recurrent neural network regularization. *arXiv preprint arXiv:1409.2329*

Zhou, Guo-Bing, Jianxin Wu, Chen-Lin Zhang, Zhi-Hua Zhou. 2016. Minimal gated unit for recurrent neural networks. *International Journal of Automation and Computing 13.3(2016)*: 226-234.